В.Н. Касьянов

ЯЗЫК ПРЕДСТАВЛЕНИЯ ГРАФОВ GRAPHML: БАЗОВЫЕ СРЕДСТВА[1]

ВВЕДЕНИЕ

Современное программирование нельзя представить себе без теоретико-графовых методов и алгоритмов [3]. Широкая применимость графов связана с тем, что они являются очень естественным средством объяснения сложных ситуаций на интуитивном уровне. Эти преимущества представления сложных структур и процессов графами становятся еще более ощутимыми при наличии хороших средств их визуализации [4, 11].

Ясно, что инструменты визуализации информации на основе графовых моделей, подобно всем другим инструментам, имеющим дело со структурированными данными, нуждаются в сохранении и передаче графов и ассоциированных с ними данных. Среди многочисленных форматов файлов для представлений графов применяются основанные на ASCII-кодах таблицы (матрицы) или списки, такие как табулированные файлы, к которым относятся *.dl файлы UCINET [5] и *.net файлы Паёка (Pajek) [10]. Есть среди них и основанные на XML форматы для представления графов, такие как GXL [17] и DyNetML [14]. Еще один используемый формат представления графов – это язык GML (Graph Modelling Language) [12], работа над которым началась в 1995 на 4-м симпозиуме по рисованию графов GD-95 в г. Пассау и завершилась в 1996 на 5-м симпозиуме по рисованию графов GD-96 в г. Беркли. Язык GML до сих остается основным файловым форматом для системы Graphlet, а также поддерживается рядом других систем обработки графов.

Однако долгое время среди используемых форматов представления графов не находилось ни одного, который был бы достаточно широко принятым в качестве стандартного; по существу, инструменты работы с графами поддерживали (да и сейчас многие из них поддерживают) лишь некоторую часть из существующих клиентских форматов, обычно состоящую из

[1] Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (грант РФФИ № 12-07-00091)

видов графовых представлений, ограниченных по выразимости и специфике конкретной областью применения.

Поэтому неслучайно в 2000 году наблюдательный комитет симпозиума по рисованию графов (Graph Drawing Steering Committee) организовал рабочее совещание по форматам обменами графовыми данными, состоявшееся в г. Вильямсбурге в рамках 8-го симпозиума по рисованию графов (GD-2000) [9]. Как следствие была сформирована неформальная рабочая группа по выработке основанного на языке XML формата обмена графами GraphML, который, в частности, был бы пригоден для обмена данными между инструментами рисования графов и другими приложениями и в конечном счете лёг бы в основу стандарта описания графов.

Рабочая группа по созданию языка GraphML объединяет десятки специалистов из разных организаций и стран, и её работу наравне с другими координируют Ulrik Brandes (Университет г. Констанц, Германия), Markus Eiglsperger (Тюбингенский университет, Германия), Michael Kaufmann (Тюбингенский университет, Германия), Jürgen Lerner (Университет г. Констанц, Германия) и Christian Pich (Университет г. Констанц, Германия).

Первый отчёт по языку вышел в 2001 году [6]. С тех пор язык был расширен в части поддержки основных типов атрибутов и в части включения информации для использования синтаксическими анализаторами [7, 8]. Ведется работа по включению абстрактной информации для описания топологии графа и шаблонов, с помощью которых эту информацию можно преобразовать в различные графические форматы. Программное обеспечение для поддержки работы с GraphML также находится в стадии разработки.

Благодаря XML синтаксису GraphML может использоваться в комбинации с другими форматами, основанными на XML. С другой стороны, свой собственный механизм расширения позволяет прикреплять <data> метки со сложным содержимым элементов GraphML, возможно, требуемым для исполнения с другими моделями XML содержимого. Примером использования таких меток со сложным содержимым является так называемый механизм SVG (Scalable Vector Graphics) [15], описывающий появление вершин и дуг в изображении. С другой стороны, GraphML может интегрироваться в другие приложения, например, в SOAP сообщения [16].

В данной статье рассматриваются базовые средства языка GraphML, достаточные для представления графовых моделей в большинстве приложений. В ней описывается, как графы и простые графовые данные представляются в формате GraphML. Другим возможностям языка, связанным с его расширением за счет введения дополнительных понятий для графовой

топологии, таких как вложенные графы, гиперграфы и порты, посвящена отдельная статья [2].

1. ЦЕЛИ РАЗРАБОТКИ И ИСПОЛЬЗУЕМАЯ ГРАФОВАЯ МОДЕЛЬ

Разработчики языка убеждены, что современный формат обмена графами не может быть монолитным, поскольку сервисы рисования графов используются в качестве компонентов более больших систем, и возникают сетевые сервисы. Всегда может потребоваться обмен графовыми данными между такими сервисами или этапами сервиса, а также между сервисами рисования графов и системами, специфическими для областей приложений.

Типичные пользовательские сценарии, которые предусматривались авторами для разработанного формата, собраны вокруг систем, спроектированных для произвольных приложений, имеющих дело с графами и другими данными, ассоциированными с ними. Такие системы могут содержать или вызывать сервисы рисования графов, которые добавляют или изменяют раскладку или графическую информацию. Такие сервисы могут вычислять только частичную информацию или промежуточные представления, поскольку они воплощают лишь часть многофазового подхода к укладке, такого как метрики топологических форм (the topology-shape-metrics) или схемы Сугиямы (Sugiyama frameworks) [10, 13].

Основную цель разработчики языка формулируют следующим образом. Формат обмена графами должен быть в состоянии представлять произвольные графы с произвольными дополнительными данными, включая укладку и графическую информацию. Дополнительная информация должна сохраняться в формате, подходящем для заданного конкретного приложения, но не должна усложнять представление данных из других приложений или мешать ему.

GraphML проектировался с ориентацией на эту цель, а также с учётом следующих более прагматических целей.

- Простота (Simplicity). Формат должен был прост для разбора и интерпретации как людьми, так и машинами. В качестве общего принципа формулируется отсутствие неоднозначностей и, таким образом, существование единственной хорошо определенной интерпретации для каждого валидного (valid) GraphML-документа.
- Общность (Generality). Не должно существовать ограничений по от-

ношению к графовой модели, т.е. гиперграфы[2], иерархические графы и т.д. должны быть выразимы с помощью одного и того же базисного формата.

- Расширяемость (Extensibility). Должна существовать возможность расширять формат хорошо определённым способом для представления дополнительных данных, требуемых произвольными приложениями или более сложным использованием (например, посылая алгоритм раскладки вместе с графом).
- Робастность (Robustness). Система, не способная обработать весь диапазон графовых моделей или дополнительной информации, должна быть в состоянии легко распознавать и извлекать то подмножество, которое она может обработать.

Под графовой моделью, описываемой с помощью базовых средств языка, понимается помеченный смешанный мультиграф

$$G = (V, E, L),$$

состоящий из множества вершин V, множества рёбер E (как неориентированных, так и ориентированных), соединяющих пары необязательно различных вершин, и множества разметок L, каждая из которых является частичной функцией, сопоставляющей элементам графа $\{G\} \cup V \cup E$ элементы некоторого заданного множества пометок.

Таким образом, рассматриваемая графовая модель включает графы, которые могут содержать ребра, дуги, петли, кратные дуги и кратные рёбра. Множества пометок могут кодировать, например, различные семантические свойства объектов, представленных в виде данной графовой модели, или различные геометрические свойства элементов графа в заданном его изображении на плоскости.

2. ЗАГОЛОВОК

В качестве примера представления рассмотрим на рис. 1 фрагмент GraphML-документа, который представляет простой непомеченный граф, изображённый на рис. 2.

[2] Здесь и ниже мы без определения используем стандартные понятия из теории графов (см., например, [1]).

```
<graphml>
        <graph edgedefault="directed">
        <node id="v1"/>
        <node id="v2"/>
        <node id="v3"/>
        <node id="v4"/>
        <edge source="v1" target="v2"/>
        <edge source="v1" target="v3"/>
        <edge source="v2" target="v4"/>
        <edge source="v2" target="v4" directed="false"/>
        </graph>
</graphml>
```

Рис.1. Фрагмент GraphML-документа для представления простого графа, изображённого на рис. 2

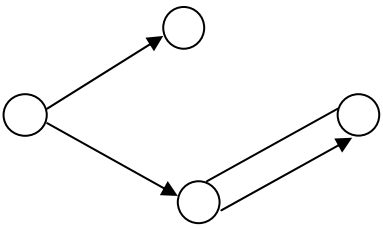

Рис. 2. Пример простого графа

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns=http://graphml.graphdrawing.org/xmlns
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
        http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
        <!--Content: List of graphs and data-->
</graphml>
```

Рис. 3. Минимальный валидный GraphML-документ

Следует отметить, что указанный фрагмент GraphML-документа как XML-документ не валиден, поскольку каждый валидный XML-документ должен декларировать в своём заголовке, является ли он DTD (document type definition) или XML-схемой. По-существу, заданное множество определений DTD и XML-схем определяет подмножество всех тех XML-документов, которые и формируют некоторый конкретный язык. Но язык GraphML был определен с помощью лишь некоторой схемы. И хотя DTD-определение предназначено для поддержки парсеров, которые не могут обработать схемные определения, единственной нормативной спецификацией языка GraphML является GraphML-схема, размещенная по адресу

http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd

Документ, представленный на рис. 3, является минимальным GraphML-документом, который валиден по данной схеме. Понятно, что данный документ определяет пустое множество графов. Части документа, начинающиеся с символов <!-- и завершающиеся символами -->, являются комментариями.

Первая строка документа – это инструкция обработки, которая определяет, что документ является подмножеством стандарта XML 1.0, и что документ выполнен в кодировке UTF-8, являющейся стандартом для XML-документов. Конечно, для GraphML-документов могут быть выбраны и другие кодировки.

Вторая строка содержит корневой элемент GraphML-документа – graphml, который, как и все остальные элементы языка GraphML, принадлежит пространству имен

http://graphml.graphdrawing.org/xmlns.

По этой причине с помощью XML-атрибута

xmlns="http://graphml.graphdrawing.org/xmlns"

именно это пространство имен в нашем документе и определено в качестве пространства имен данного документа, заданного по умолчанию. Следующие два XML-атрибута определяют XML-схему, которая используется для валидации данного документа. В нашем примере используется стандартная схема GraphML-документа, расположенная на сервере graphdrawing.org. Первый атрибут,

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance",

определяет xsi в качестве префикса пространства имен XML-схемы. Второй атрибут,

xsi: schemaLocation="http://graphml.graphdrawing.org/xmlns
http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd",

определяет местонахождение XML-схемы для элементов пространства имен GraphML. Он предоставляет информацию о том, что все элементы в пространстве имен GraphML являются валидными по отношению к файлу graphml.xsd, расположенному по указанному адресу. Конечно, валидация не должна обязательно выполняться с использованием данного файла. Локальные копии graphml.xsd также могут специфицироваться в качестве местонахождений XML-схем. Заметим, что обычно значение атрибута schemaLocation является списком пар, в которых первый элемент обозначает некоторое пространство имен, а второй указывает на файл, в котором элементы этого пространства определены.

Ссылка на XML-схему не обязательна, но она обеспечивает механизм для синтаксической проверки документа и поэтому строго рекомендуется. Минимальный GraphML-документ без ссылки на схему приведен на рис. 4. Заметим, что этот файл не является валидным документом в соответствии с XML-спецификацией.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml
xmlns="http://graphml.graphdrawing.org/xmlns" >
        <!--Content: List of graphs and data-->
</graphml>
```

Рис. 4. Минимальный GraphML-документ без ссылки на схему

3. ТОПОЛОГИЯ

Опишем как топология графа (его элементы) представляется на языке GraphML.

Вначале еще раз рассмотрим документ, приведенный на рис. 1. Отдельный граф представлен на языке GraphML посредством элемента <graph>. Элемент <graphml> может содержать произвольное число элементов <graph>.

Каждый граф в GraphML может являться смешанным, другими словами, он может содержать одновременно как ориентированные, так и неориенти-

рованные ребра. Если при объявлении ребра его ориентированность не определена, то применяется ориентированность, заданная для ребер графа по умолчанию. Ориентированность ребер графа, присваиваемая по умолчанию, задается с помощью XML-атрибута edgedefault элемента <graph>. Данный XML-атрибут может принимать одно из двух значений: directed (ориентированный) и undirected (неориентированный). Значение по умолчанию должно быть задано обязательно. Дополнительно с помощью атрибута id, графу может быть присвоен некоторый идентификатор. Этот идентификатор нужен тогда, когда на данный граф требуется организовать ссылку.

Вершины графа представляются в виде списка элементов <node>. Каждая вершина имеет уникальный в пределах данного документа идентификатор, который задается с помощью атрибута id.

Множество ребер представляется в виде списка элементов <edge>. Каждое ребро имеет две инцидентные вершины, задаваемые с помощью XML-атрибутов source и target. Значения атрибутов source и target должны содержать идентификаторы вершин, определенных в том же документе что и ребро. Ребра с одной инцидентной вершиной, так называемые петли, определяются с помощью одинаковых значений, заданных в атрибутах source и target. Дополнительный XML-атрибут directed определяет ориентированность ребра, заданную в явном виде. Значение true задает ориентированное ребро, а false – неориентированное. Если ориентированность в явном виде не задана, то применяется ориентированность, заданная по умолчанию при объявлении графа. Дополнительно с помощью XML-атрибута id может быть задан идентификатор ребра. XML-атрибут id задается, когда необходимо организовать ссылку на данное ребро.

Вершины и ребра упорядочиваются произвольным образом, и язык не требует перечислять все вершины до перечисления всех рёбер. Ясно, что память, требуемая для сохранения на языке GraphML графа с n вершинами и m рёбрами, составляет $O(n + m)$.

4. АТРИБУТЫ

В предыдущем разделе мы обсудили порядок описания топологии графа на языке GraphML. Хотя имеется целый ряд приложений, для которых информации о топологии графов может быть достаточно, большинство приложений работает с графовыми моделями, обладающими дополнительной

информацией. Поэтому в языке GraphML предусмотрены средства для включения различной информации в описание графа.

С помощью механизма расширения, который называется *GraphML-атрибуты,* для элементов графа может быть задана дополнительная информация простого типа. Простой тип подразумевает, что данные ограничены скалярными величинами, например, числами и строками. Расширение GraphML-атрибуты уже включено в файл

http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd

и таким образом заголовок графа со скалярными атрибутами может иметь вид, рассмотренный в разд. 2.

GraphML-атрибуты не следует путать с XML-атрибутами, которые имеют совсем другой смысл. GraphML-атрибуты добавляют информацию к графам, множествам графов или частям графов, а XML-атрибуты добавляют информацию к XML-элементам.

В большинстве случаев дополнительная информация может и должна прикрепляться к элементам GraphML с помощью механизма GraphML-атрибутов, описанного здесь. Это гарантирует читаемость описаний графов другими GraphML-парсерами. Если же необходим более сложный формат данных в качестве атрибутов, можно воспользоваться механизмом расширения языка GraphML произвольными данными в четко определённых местах. Осуществление таких расширений мы описали в другой статье [2].

GraphML-атрибуты рассматриваются как частичные функции, приписывающие элементам графа значения атрибутов, которые, как правило, имеют один и тот же тип. Например, веса ребер могут рассматриваться как функция из множества ребер E в множество вещественных чисел R:

$$\text{weight}: E \to R .$$

Другой пример – это формы изображения вершин, которые можно представить в виде функции из множества вершин V в множество слов над заданным алфавитом Σ:

$$\text{shape}: V \to \Sigma^* .$$

Для добавления указанных функций к элементам графа следует использовать key/data-механизм языка GraphML. Элемент <key>, размещаемый в начале документа, декларирует новую функцию разметки; более точно элемент <key> специфицирует для функции её идентификатор, имя, области

определения и значения. Сами же значения функции определяются с помощью <data> элементов.

Декларация всех функций разметки в самом начале документа позволяет парсерам построить подходящие структуры данных в начале процесса разбора. Также парсеры могут распознавать ситуации, когда необходимые данные опущены. На рис. 5 приведен пример применения key/data-механизма. Здесь функция веса описывается строкой

<key id="d1" for="edge" attr.name="weight" attr.type="double"/>.

Элемент <key> имеет XML-атрибут с именем for, который специфицирует область определения функции. Для атрибута for в качестве значений можно указать graph, node, edge, graphml, а также имена других типов элементов графа, рассмотренные в разд. 2. Данный XML-атрибут может также получить значение all, что означает, что данные пометки могут помечать любые элементы графа. Атрибут for вместе с уникальным атрибутом id являются обязательными для элементов <key>. Расширение GraphML предоставляет еще два атрибута для <key>: это атрибут attr.name, который определяет имя функции и используется парсером для распознавания соответствующих данных, а также атрибут attr.type, который задает область значения функции и может принимать в качестве значений имена типов[3] boolean, int, long, float, double, или string.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
<key id="d0" for="node" attr.name="shape" attr.type="string">
<default>circle </default>
</key>
<key id="d1" for="edge" attr.name="weight" attr.type="double"/>
<graph id="G" edgedefault="undirected">
<node id="n0">
<data key="d0">square</data>
</node>
<node id="n1"/>
<node id="n2">
<data key="d0">oval</data>
</node>
<node id="n3">
```

[3] Эти типы определены в соответствии с аналогичными типами в языке Java.

```
<data key="d0">square</data>
</node>
<node id="n4"/>
<node id="n5">
<data key="d0">oval</data>
</node>
<node id="n6"/>
<edge id="e0" source="n0" target="n2">
<data key="d1">1.0</data>
</edge>
<edge id="e1" source="n0" target="n1">
<data key="d1">1.0</data>
</edge>
<edge id="e2" source="n1" target="n3">
<data key="d1">2.0</data>
</edge>
<edge id="e3" source="n3" target="n2"/>
<edge id="e4" source="n2" target="n4"/>
<edge id="e5" source="n3" target="n5"/>
<edge id="e6" source="n5" target="n4">
<data key="d1">1.1</data>
</edge>
</graph>
</graphml>
```

Рис. 5. Представление атрибутированного графа, в котором ребра имеют веса, а вершины форму

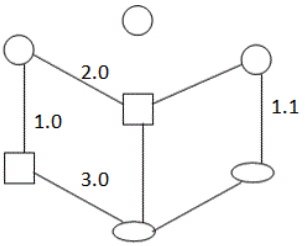

Рис. 6. Изображение атрибутированного графа, представленного в виде GraphML-документа на рис. 5

Обычный парсер, обрабатывая веса ребер, как правило, после обработки описания функции веса инициализирует внутреннюю структуру данных, которая сохраняет двойное вещественное с каждым ребром. В отличие от него другой парсер, который не знает о функции пометок ребер весами или не нуждается в этой функции, будет просто игнорировать ассоциированные <data> элементы.

Значения функции данных на некотором элементе графа (или, что то же самое, значение GraphML-атрибута у данного элемента графа) задается с помощью элемента <data>, вложенного в описание данного элемента. Например, фрагмент документа

```
<edge id="e0" source="n0" target="n2">
<data key="d1">1.0</data>
</edge>
```

определяет значение 1.0 в качестве веса для заданной дуги <edge>. Элемент data имеет XML-атрибут <key>, который ссылается на идентификатор GraphML-атрибута. Значение GraphML-атрибута задается текстовым содержимым элемента <data>. Это значение должно иметь тип, объявленный в соответствующем элементе <key>.

Для GraphML-атрибутов можно определить значение по умолчанию. Содержимое элемента default определяет текстовое значение по умолчанию. Например, фрагмент документа

```
<key id="d0" for="node" attr.name="shape" attr.type="string">
<default>circle </default>
</key>
```

определяет значение circle в качестве значения по умолчанию для атрибута форма у вершин графа.

Могут быть такие GraphML-атрибуты, которые определены, но не объявлены с помощью элемента <data>. Если значение по умолчанию определено для данного GraphML-атрибута, то тогда это значение применяется к соответствующему (входящему в домен GraphML-атрибута) элементу графа. В вышеприведенном примере значение не определено для вершины с идентификатором n1 и GraphML-атрибута с именем shape. Однако для данного GraphML-атрибута определено значение по умолчанию circle, которое будет присвоено данной вершине. Если же значение по умолчанию не задано, как это имеет место для GraphML-атрибута weight в вышеприведенном

примере, то значение GraphML-атрибута для такого элемента графа считается неопределенным. В вышеприведенном примере не определено значение GraphML-атрибута, задающего вес, у ребра с идентификатором e3.

5. ИНФОРМАЦИЯ ДЛЯ ПАРСЕРА

Помимо возможности задания атрибутов к базовым средствам языка относят еще одно расширение формата описания графа, называемое GraphML-Parseinfo. GraphML-Parseinfo делает возможным писать простые парсеры, основанные на дополнительной информации в GraphML-файлах. Это расширение уже включено в файл:

http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd

Таким образом, заголовок файла с описанием графа, использующим это расширение, может иметь тот же вид, что и рассмотренный ранее (в разд. 2).

Указанное расширение направлено на оптимизацию синтаксического разбора документа с помощью парсера за счет использования специальных метаданных, которые могут быть добавлены к некоторым GraphML-элементам с помощью XML-атрибутов. Имеется два вида метаданных: информация о количестве элементов и информация о способе кодирования конкретных данных в документе. Например, для парсера, который сохраняет вершины и инцидентные ребра в форме массивов, может оказаться весьма полезной информация о количествах вершин и ребер в графе, а также степенях его вершин. Все XML-атрибуты, задающие указанные метаданные, имеют префикс parse.

Для первого вида метаданных, связанного с информацией о количеством элементов, определены следующие XML-атрибуты для элемента <graph>: XML-атрибут parse.nodes задает количество вершин в графе, XML-атрибут parse.edges определяет количество ребер в графе, XML-атрибут parse.maxindegree определяет максимальное количество ребер, входящих в одну вершину графа, а XML-атрибут parse.maxoutdegree определяет максимальное количество ребер, исходящих из одной вершины графа. Кроме того, для элемента <node> введены новые XML-атрибуты parse.indegree и parse.outdegree, котрые определяют для данной вершины количества входящих и исходящих ребер.

Для метаданных, связанных со способом кодирования, определены XML-атрибуты parse.nodeids, parse.edgeids и parse.order для элемента

<graph> со следующей семантикой. Если XML-атрибут parse.nodeids имеет значение canonical, все вершины получают идентификатор вида nX, где X обозначает количество элементов <node>, предшествующих данному элементу. Другое возможное значение данного XML-атрибута равно free. Аналогичным образом действует XML-атрибут parse.edgeids, который задает вид идентификатора для узлов. Отличие состоит только в том, что этот идентификатор имеет вид eX. XML-атрибут parse.order определяет порядок, в котором узлы и ребра располагаются в документе. При значении, равном nodesfirst, все элементы <node> располагаются раньше элементов <edge>. При значении, равном adjacencylist, объявление вершины предшествует объявлению смежных ей вершин. Для значения free порядок следования вершин и ребер никак не ограничивается.

Пример на рис. 7 иллюстрирует использование указанного вида информации для парсера.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
<graph id="G" edgedefault="directed"
                parse.nodes="11"
                parse.edges="12"
                parse.maxindegree="2"
                parse.maxoutdegree="3"
                parse.nodeids="canonical"
                parse.edgeids="free"
                parse.order="nodesfirst">
        <node    id="n0"   parse.indegree="0"     parse.outdegree="1"/>
        <node    id="n1"   parse.indegree="0"     parse.outdegree="1"/>
        <node    id="n2"   parse.indegree="2"     parse.outdegree="1"/>
        <node    id="n3"   parse.indegree="1"     parse.outdegree="2"/>
        <node    id="n4"   parse.indegree="1"     parse.outdegree="1"/>
        <node    id="n5"   parse.indegree="2"     parse.outdegree="1"/>
        <node    id="n6"   parse.indegree="1"     parse.outdegree="2"/>
        <node    id="n7"   parse.indegree="2"     parse.outdegree="0"/>
        <node    id="n8"   parse.indegree="1"     parse.outdegree="3"/>
        <node    id="n9"   parse.indegree="1"     parse.outdegree="0"/>
        <node    id="n10"  parse.indegree="1"     parse.outdegree="0"/>
        <edge     id="edge0001"     source="n0"    target="n2"/>
        <edge     id="edge0002"     source="n1"    target="n2"/>
        <edge     id="edge0003"     source="n2"    target="n3"/>
```

```
        <edge    id="edge0004"    source="n3"    target="n5"/>
        <edge    id="edge0005"    source="n3"    target="n4"/>
        <edge    id="edge0006"    source="n4"    target="n6"/>
        <edge    id="edge0007"    source="n6"    target="n5"/>
        <edge    id="edge0008"    source="n5"    target="n7"/>
        <edge    id="edge0009"    source="n6"    target="n8"/>
        <edge    id="edge0010"    source="n8"    target="n7"/>
        <edge    id="edge0011"    source="n8"    target="n9"/>
        <edge    id="edge0012"    source="n8"    target="n10"/>
</graph>
</graphml>
```

Рис. 7. Использование GraphML-Parseinfo метаданных

СПИСОК ЛИТЕРАТУРЫ

1. Евстигнеев В. А., Касьянов В. Н. Толковый словарь по теории графов в информатике и программировании. - Новосибирск: Наука, 1999.
2. Касьянов В.Н. Язык представления графов GraphML: дополнительные возможности // Информатика в науке и образовании. – Новосибирск, 2012. – С. 23–46.
3. Касьянов В. Н., Евстигнеев В. А. Графы в программировании: обработка, визуализация и применение. ─ СПб.: БХВ-Петербург, 2003.
4. Касьянов В. Н., Касьянова Е. В. Визуализация графов и графовых моделей. — Новосибирск: Сибирское Научное Издательство, 2010.
5. Borgatti S.P., Everett M.G., and Freeman L.C. UCINET 6.0 / Analytic Technologies, 1999.
6. Brandes U. GraphML progress report: structural layer proposal / Brandes U., Eiglsperger M. et al. / Lecture Notes in Comput. Sci. ─2002. ─Vol. 2265. ─ P. 501–512. ─ (Proc. 9th Int. Symp. Graph Drawing GD'2001).
7. Brandes U., Eiglsperger M., Lerner J. GraphML Primer. http://graphml.graphdrawing.org/primer/graphml-primer.html#EXT
8. Brandes U. Graph markup language (GraphML)/ Brandes U., Eiglsperger M. et al. http://www.cs.brown.edu/~rt/gdhandbook/chapters/graphml.pdf
9. Brandes U.. Graph data format workshop report / Brandes U., Marshall M.S., North S.C. / Lecture Notes in Comput. Sci. – 2001. -Vol. 1984. – P. 410 – 418. - (Proc. 8th Int. Symp. Graph Drawing GD'2000).
10. De Nooy W., Mrvar A., and Batagelj V. Exploratory social network analysis with Pajek. - Cambridge University Press, 2005.
11. Di Battista G. et al. Graph Drawing: Algorithms for the Visualization of Graphs/ Di Battista G., Eades P., Tamassia R., Tollis I.G. - Prentice Hall, 1999.

12. GML. The Graph Modeling Language File Format. http://www.infosun.fmi.uni-passau.de/Graphlet/GML/.
13. Sugiyama K. Methods for visual understanding of hierarchical system structures / Sugiyama K., Tagawa S., Toda M / IEEE Transactions on Systems, Man and Cybernetics. ─1981. ─Vol. 11, N 2. ─P. 109–125.
14. Tsvetovat M., Reminga J., and Carley K. Dynetml: interchange format for rich social network data / NAACSOS Conference. - Pittsburgh, PA, 2003.
15. W3C. Scalable Vector Graphics. http://www.w3.org/TR/SVG/.
16. W3C. SOAP. http://www.w3.org/TR/soap12-part0/.
17. Winter A. Exchanging Graphs with GXL / Lecture Notes in Comput. Sci. ─2002. ─ Vol. 2265. ─P. 485–500. ─ (Proc. 9th Int. Symp. Graph Drawing GD'2001).